Homework 3 : xv6 system call

2017030519 컴퓨터소프트웨어학부 홍유진

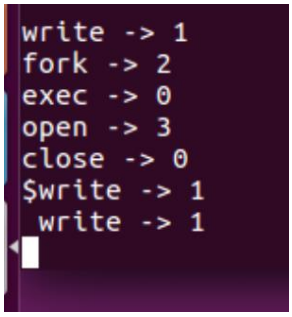
1. system call tracing

```
void
syscall(void)
{
  int num;
  struct proc *curproc = myproc();

  num = curproc->tf->eax;
  if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
    curproc->tf->eax = syscalls[num]();
    cprintf("%s -> %d\n",name[num],curproc->tf->eax);

} else {
    cprintf("%d %s: unknown sys call %d\n",
            curproc->pid, curproc->name, num);
    curproc->tf->eax = -1;
  }
}
```

- Syscall.c의 수정한 syscall()함수
- name배열을 통해 num에 인덱스에 있는 시스템콜을 출력할 수 있도록함

```
static char *name[] = {
[SYS_fork]    "fork",
[SYS_exit]    "exit",
[SYS_wait]    "wait",
[SYS_pipe]    "pipe",
[SYS_read]    "read",
[SYS_kill]    "kill",
[SYS_exec]    "exec",
[SYS_fstat]   "fstat",
[SYS_chdir]   "chdir",
[SYS_dup]     "dup",
[SYS_getpid]  "getpid",
[SYS_sbrk]    "sbrk",
[SYS_sleep]   "sleep",
[SYS_uptime]  "uptime",
[SYS_open]    "open",
[SYS_write]   "write",
[SYS_mknod]   "mknod",
[SYS_unlink]  "unlink",
[SYS_link]    "link",
[SYS_mkdir]   "mkdir",
[SYS_close]   "close",
};
```

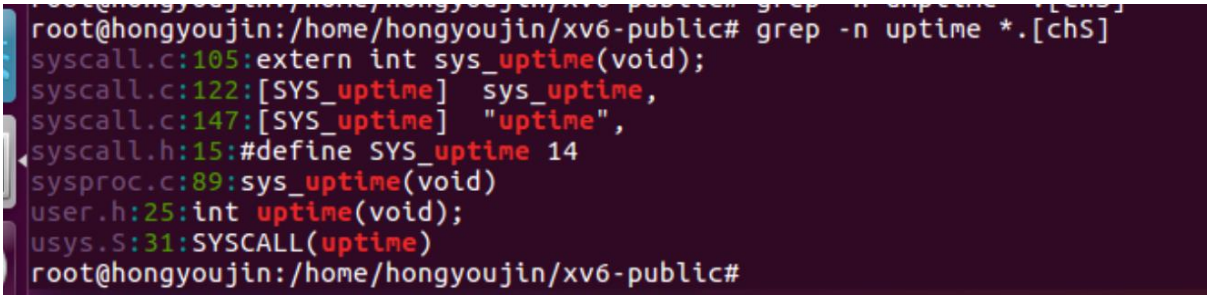
- name배열



- Xv6 실행화면

2. Date system call

- Makefile에 정의 되어있는 UPROGS에 _date₩를 추가했다. 이를 통해 컴파일러가 이와 같은 프로그램을 가지고 있으며 다른 시스템 프로그램과 함께 컴파일 해야한다고 알려준다.//make할 때 user program code를 build한다.



- grep -n uptime *.[chS]명령어를 통해 uptime을 사용하는 코드를 볼 때, uptime시스템콜은 시스템콜을 하기위해 1. syscall.c 2. syscall.h 3. sysproc.c 4.user.h 5. usys.S 파일에서 사용된 것을 알 수 있다.

1) syscall.c

```
extern int sys_chdir(void);
extern int sys_close(void);
extern int sys_dup(void);
extern int sys_exec(void);
extern int sys_exit(void);
extern int sys_fork(void);
extern int sys_fstat(void);
extern int sys_getpid(void);
extern int sys_kill(void);
extern int sys_link(void);
extern int sys_mkdir(void);
extern int sys_mknod(void);
extern int sys_open(void);
extern int sys_pipe(void);
extern int sys_read(void);
extern int sys_sbrk(void);
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_date(void);
```

- syscall.c에서 extern int sys_date(void)를 추가했다.

```
static int (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_kill]    sys_kill,
[SYS_exec]    sys_exec,
[SYS_fstat]   sys_fstat,
[SYS_chdir]   sys_chdir,
[SYS_dup]     sys_dup,
[SYS_getpid]  sys_getpid,
[SYS_sbrk]    sys_sbrk,
[SYS_sleep]   sys_sleep,
[SYS_uptime]  sys_uptime,
[SYS_open]    sys_open,
[SYS_write]   sys_write,
[SYS_mknod]   sys_mknod,
[SYS_unlink]  sys_unlink,
[SYS_link]    sys_link,
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
[SYS_date]    sys_date,
};
```

- syscall.c에서 static int (*syscalls[])(void) 함수포인터의 마지막에 [SYS-date] sys_date, 를 추가했다. 이를 통해 시스템콜 번호에 해당하는 배열의 entry에 시스템콜 핸들러의 주소를 저장한다.

2) syscall.h

```
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
#define SYS_mknod  17
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
#define SYS_date   22
"syscall.h" 23L, 507C                                  14,1          모두
```

- syscall.h에 마지막에 #define SYS_date 22를 추가했다. 이는 syscall 함수에서 시스템콜을 식별하기 위한 식별자로 사용된다.

3) sysproc.c

```
int
sys_date(void){
        char *r;
        if(argptr(0,&r,sizeof(struct rtcdate* )) < 0){
        return -1;
        }
        cmostime((struct rtcdate *)r);
        return 0;
}
"sysproc.c" 107L, 1333C
```

- sysproc.c에 int sys_date(void)함수와 그에 대한 정의를 했다. argptr함수는 사용하려고 하는 메모리의 size가 유효한지 검사하기 위해 사용하고 유효하지 않다면 -1을 반환하기 때문에 예외처리를 해준다. 또한 매개변수의 주소를 r에 저장해준다. cmostime함수를 통해 실제 시간을 읽어온다.

4) user.h

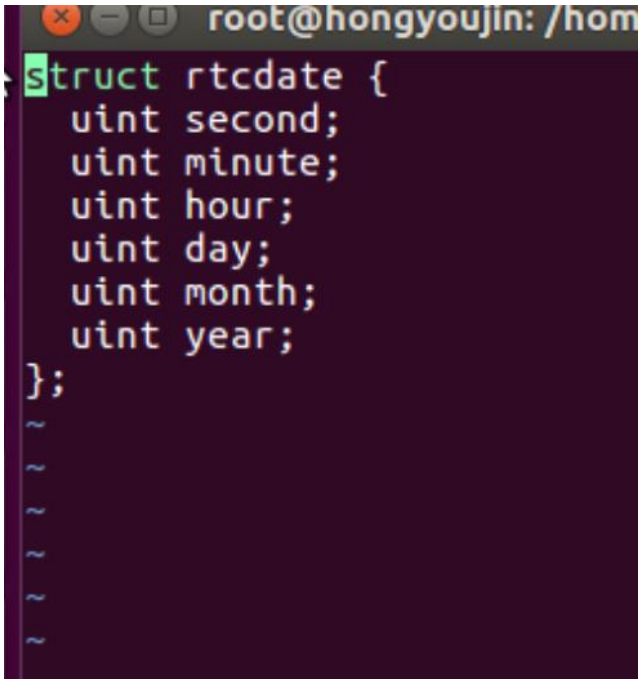
```
// system calls
int fork(void);
int exit(void) __attribute__((noreturn));
int wait(void);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int date(struct rtcdate *r);
```

int date(struct rtcdate *r);을 추가했다. 이는 user program에서 system call을 볼 수 있게해준다.

5) usys.S

```
#include "syscall.h"
#include "traps.h"

#define SYSCALL(name) \
  .globl name; \
  name: \
    movl $SYS_ ## name, %eax; \
    int $T_SYSCALL; \
    ret

SYSCALL(fork)
SYSCALL(exit)
SYSCALL(wait)
SYSCALL(pipe)
SYSCALL(read)
SYSCALL(write)
SYSCALL(close)
SYSCALL(kill)
SYSCALL(exec)
SYSCALL(open)
SYSCALL(mknod)
SYSCALL(unlink)
SYSCALL(fstat)
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(date)
~
~
```

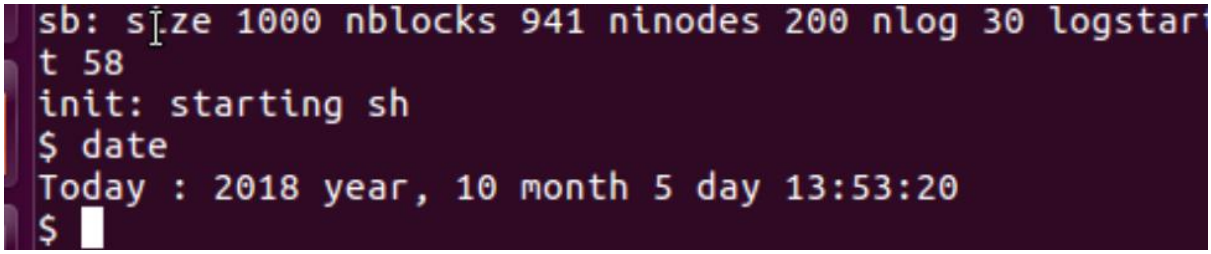
- SYSCALL(date)를 추가했다. usys.S는 eax레지스터에 시스템 콜 번호를 저장 한 수 int instruction을 수행시킨다.



- rtcdate struct를 통해 어떤 변수를 사용할 수 있는지 확인한다.

```
#include "types.h"
#include "user.h"
#include "date.h"

int
main(int argc, char *argv[]){
        struct rtcdate r;

        if (date(&r)){
        printf(2,"date failed\n");
        exit();
        }
        printf(1, "Today : %d year, %d month %d day %d:%d:%d \n",r.year,r.month,
r.day,r.hour,r.minute,r.second);

        exit();
}
```

- 출력부분을 rtcdate struct의 변수를 활용해 수정했다.



- date시스템콜의 수행결과이다.